Matlab Interface
For
Data Translation
Mach Frame grabbers

User Manual & Technical Guide

Last updated: July 2002

D.J. Oei, A. Kapoerchan, Ó1998 Prague, Czech Technical University, Czech Republic.
Modified by: R. Heil, C. L. Wauters, June 2000
Modified by: M.A.E.Bakker, L.I.Oei, July 2002

Matlab Interface For
Data Translation Mach Frame grabbers
User Manual & Technical Guide

DJ Oei (D.J.Oei@twi.tudelft.nl)
A Kapoerchan (A.Kapoerchan@twi.tudelft.nl)
August 18, 1998

Modified by:
R. Heil (R.Heil@twi.tudelft.nl)
C.L.Wauters (C.L.Wauters@twi.tudelft.nl)
June, 2000

Modified by:
M.A.E.Bakker (maarten@panic.et.tudelft.nl)
L.I.Oei (L.I.Oei@twi.tudelft.nl)
July, 2002

D.J. Oei, A. Kapoerchan, Ó1998 Prague, Czech Technical University, Czech Republic.
Modified by: R. Heil, C. L. Wauters, June 2000
Modified by: M.A.E.Bakker, L.I.Oei, July 2002

Contents

Appendices

- Table 1: Description of the parameters of the device info structure
- Table 2: Possible values of the parameters and their availability to set/get
- Scheme Matlab interface for DT Open Layer Frame grabbers

PART I: User Manual

1. Introduction

This document describes the Matlab interface for DT Open Layer Frame grabbers. It will describe the functionality of this interface and under which conditions the interface has to work. This document can be used as reference while using this interface.

1.1 General description

DT Open Layer Frame grabbers are used for video capturing. You can connect several cameras to this frame grabber depending on the number of input channels. With the frame grabber software development kit you can make your own program to grab an image with a connected camera for example. You can even grab a sequence of images. It then will be a live image. More info about frame grabbers will be given in chapter 2.

This grabbing of an image with a frame grabber will now be done via Matlab. The aim of this document is to explain and describe a Matlab Interface for DT Open Layer Frame grabbers. With this interface you can control an installed frame grabber within Matlab. The interface will be split into several DLL's, the executable for Mex-files in Matlab. The code is made mostly in C but some special Mex-functions from Matlab are used too. To control the frame grabber, functions from the functional library of the frame grabber software development kit will be extensively used.
This interface is made for use by people of the Center of Machine Perception (CMP), which is a department of the faculty of Electrical Engineering of the Czech Technical University in Prague.

1.2 Brief Matlab description

Matlab is a mathematical program that is very popular at the universities all around the world. You can compute many things with it and you can even make application interfaces with Matlab. The interface is made with Mex-files. After compiling and linking these files, Matlab creates DLL's (Dynamic Link Libraries), which can be executed within Matlab. The language of these Mex-files is C. But Matlab itself has also some special Mex-functions that can be used in these Mex-files. For further reference, check the Matlab manuals and books.

1.3 Hardware requirements

The interface is developed under Matlab 5.3.1. The Mex compiler is used to create the DLL's. At least Matlab 5 for W95/NT and a DT frame grabber is needed to work with this interface. A camera connected to the frame grabber should be very useful too.

2 About Frame grabbers

Frame grabbers are Open Layer Devices. DT-Open Layers is a set of open standards for developing integrated modular software under Microsoft Windows and hardware independent support.

One of the main advantages of DT-Open Layers is that when you have written a program you can use it with new boards too. There's no need to rewrite your code. The reason for this is that the software uses similar device calls and memory management between modules. Therefore it's easy to put some new function into an existing product. You can add your new functions by simply adding a function library. You can do this by including additional DT-Open Layer compliant function libraries. This makes it enormous flexible because you can combine as much software libraries and corresponding supported hardware, as you like. The functions can be written in a higher-level language such as C and Visual Basic.

DT-Open Layers uses 3 standard Layers:
First there's the Application Layer. It represents the Windows application software for the end users. The Application Layer is connected to the Function Library Layer. This connection is called the Application Programming Interface (API); function calls that applications can use to interface with function libraries.
The function-specific programmer's libraries (DLL) (like DataAcq SDK and frame grabber SDK) are used to assist the OEMs, system integrators, and developers. The Function Library Layer is connected to the Device Interface Layer. This connection is called the System Programming Interface (SPI).
The Device Interface Layer (DLL), consisting of lower-level device drivers called by the function libraries are used to assert control over specific devices. It provides a device-independent interface for the native OS. So the common part of all frame grabbers is controllable

Beneath the Device Interface Layer are the boards. Because DT-Open Layers uses the Windows Dynamic Link Library (DLL) feature for its function libraries and device drivers, you can generally update or replace your board without recompiling. Because it is device-independent, you must query the DT-Open Layer to find out what it can and can't do. With this information you can tell the user that some action is allowed or not. There are basic functions described in the FG SDK manual. With these functions you are able to write your own code to control the frame grabber.

The capability of the frame grabber

The basic thing is to get the device open, do your work and close the device. It isn't that easy. You must first identify a DT-Open Layer device before any work can be done. So what are the most basic things that all DT-open Layer devices will be able to do?

The basic steps are:

- Get the number of DT-Open Layers-compliant devices in the system;
- Get the device information (like alias);
- Open each device;
- Determine the capabilities for each opened device;
- Determine the controls for each supported device capability;
- Program the supported controls as necessary;
- Use the device and close it when finished.

You can write an easy demo. Because all the function are given you are able to make your own demo work. The names they are using are very long but the function's name gives a general idea about its functionality.

Memory usage

DT-Open Layers devices use two memory types: volatile and non volatile.

Volatile memory is a pre-defined memory area that can be changed.

Non-volatile memory is a pre-defined memory area too, but the difference is that the user can only change it. No one else may change it. The reason why volatile memory is available is that you might need a lot of memory and the best way is then to overwrite your original memory area. Another reason is the use of non-standard input devices. Then to allow the format or dimensions of the memory to be altered for any given acquisition. Once the format of the data is altered, the data is no longer valid.

There are 2 ways to store an image: store it in the host memory or in the device memory. Host memory is the available physical memory in your system.

Device memory is a contiguous block of host memory that is allocated to the device driver during system start-up. The amount you want to allocate is done when you install the drivers.

To explain what can be done with memory (just a raw idea):

- Allocate memory for a frame to store in;
- And of course destroy the allocated memory;
- Ask information about frame;
- Ask for memory capabilities;
- Some pixel works (write, read).

It is also important to know how the frame is grabbed.

You must also know a few things before grabbing some words:

- Video Signal
- Active Video Area
- Frame

How these things work are good to know. It will give you an inside idea about the working of the frame grabber.

Some actions you can do with acquire:

- It is possible to grab an image synchronically or a-synchronically (in host and device memory). It is also possible to grab multiple frames synchronically or a-synchronically (but only in device memory).
- Ask to find out how the job is ready or not.
- Ask for input capabilities and set them.
- Ask for settings of the video source and some other things.
- Cancel a job (acquisition of a frame).
- How should the frame be triggered?

Camera

We use the PULNIX cameras. The camera's control consists of digital output signals or generating camera control pulses.

The commands you can issue are:

- Let the pulse generator give a pulse.
- You can ask for the camera's control capabilities and characteristics of the frame and the device driver.
- You can set the digital output masks for the camera.

Five types of DT-Open Layer Devices

There are five types of DT-Open Layer devices, which we had to work with:

Data Translation 3152 (DT3152): B/W variable scan frame grabber.
(Accommodates also analog cameras producing non-TV formats.)
Data Translation 3153 (DT3153): Color frame grabber for PAL and NTSC format.
Data Translation 3155 (DT3155): B/W TV format frame grabber (CCIR and RS-170).
Data Translation 3157 (DT3157): Frame grabber for digital cameras.
Data Translation 3133 (DT3133): Color Frame grabber for digital cameras.

3 Matlab types

The Matlab interface uses several Matlab types. These types will be briefly explained in this chapter.

- strings

The purpose of string being in MATLAB is for string handling. It can be seen as a vector whose components are the numeric codes for the characters (the first 127 codes are ASCII).

It also can be used to convert an array that contains numerical codes into a Matlab character array.

- double

Purpose Convert to double precision Double(x) will give the double precision value for X. If X is already a double precision array, double has no effect. The range of a double is from 1.7E–308 to 1.7E+308.

- char

Purpose Create character array (string). It will convert the array X that contains positive integers representing character codes into a array in Matlab (the first 127 are ASCII). There is also a bound. When you want to convert a character array into its numerical code you can use double. Or char(C) when C is a cell array of strings, it will place each element of C into the rows of the character arrays.

- uint8

When using i = uint8(x) x will be converted to an unsigned 8-bit integer. As an example x can be double. Uint8 ranges from 0 to 255. When converting and the answer is outside the specified range, the answer is not defined. When converting a type and the type is already uint8 then nothing will happen.

- int

Int stands for integer. So the compiler will know that the variable declared int is integer. Integer type is the type of natural numbers. It starts from –2147483648 to + 2147483647. There is a limit of the size of the number. When exiting this limit there will be an overflow and can give an incorrect answer.

- struct

This is used to create a structure array in Matlab. Meaning s = struct ('field1', values1,'field2', values2...). This will create a structure array with fields and values. Value 1 ,……., are cell arrays with the same size or scalar cells. So when called it will first fill the fieldname and then the value.

- mxArray

Matlab uses arrays for many data types like including scalars, vectors, matrices, strings, cell arrays, structures, and objects. All of these variables are stored as Matlab arrays. When programming in C the Matlab array is declared of the type mxArray. All MATLAB data is stored column wise.

4 How to use this interface

The interface consists of 5 basic functions:

- **openfg**
  This will open the frame grabber device, or re-get the handle when already open.
- **closefg**
  Close an opened frame grabber device.
- **grabfg**
  Grab an image from the frame grabber.
- **getfg**
  Get a parameter from the frame grabber.
- **setfg**
  Set a parameter from of the frame grabber.

And there are 4 advanced functions available:

- **resetfg**
  Reset an opened frame grabber to its initial state.
- **loadfg**
  Load settings for the frame grabber from a file.
- **isopenfg**
  Returns a double indicating whether the frame grabber has been opened or not
- **gethandlefg**
  Returns the handle of the opened frame grabber device as a device information structure

We created one internal function (*mainfg*, June 2000). This function will be used internally by the other function. For further explanation see the technical guide.

All the functions will be explained in chapter 6 after the description of the used types and structures.

The functions can be executed in the Matlab Command Window. With each function there's a M-File included with the same name of the function.
By typing for example `help openfg,` information about that function will be shown in the Command Window of Matlab.

The 19 files (10 DLL's & 9 M-files) should be put together in one directory. Then if you're in the Matlab Command Window, you should go to that specific directory before you can execute these files.
An other way to run these is to use the Matlab Command: `addpath 'path'`. This command will add the specified path, where these functions are stored, in the Matlab Search Path Definition so you can execute the functions from any directory you want.

We will open the frame grabber DT3155 and grab 50 images, set the black level and the white level (see table 1 for description of these 2 parameters) and show these 50 images in Matlab. You can create a M-File to do this.

The M-File would look like this:

```
**********************************************************************************************

% open framegrabber DT3155
m = openfg('DT3155');

% set the black level
setfg(m,'BlackLevel',10000);

% set the white level
setfg(m,'WhiteLevel',600000);

% grab 50 images and show these in Matlab
for i = 1:50
  im=grabfg(m);
  image(im');
  drawnow;
end

% close the framegrabber
closefg(m);
% or
closefg('DT3155')

**********************************************************************************************
```

code example 1.

In the next two chapters the types, structures and the functions will be explained. After these chapters you should be able to use this interface.

5 The data types

The following types and structures are used by this interface:

- **Alias**
  String with the name of a frame grabber device.
- **Filename**
  String with the full path and filename of the setting file.
- **Imgmatrix**
  The grabbed image will be stored in a matrix called mxArray.
- **Moviematrix**

  A row matrix in which an image in the Matlab movie format can be stored.
- **Parname**
  String with the name of a parameter of the frame grabber device.
- **Parvalue**
  Numeric value, where the type depends on the parameter name.
- **Devinfo**
  Matlab structure with a subset of the settings of the frame grabber device.

- **The other fields of this structure are:**
  - DevId
  - Alias
  - InputChannel
  - PixelDepth
  - TimeOut
  - InputFilter
  - FrameType
  - FrameTop
  - FrameLeft
  - FrameWidth
  - FrameHeight
  - HorFrameInc
  - VerFrameInc
  - TotPix
  - BackPorch
  - ClampStart
  - ClampEnd
  - FirstPix
  - PixCount
  - TotalLines
  - FirstLine
  - LineCount

- VideoType
- SyncTresHold
- SyncSource
- LineLo2Hi
- FieldLo2Hi
- SyncSentinel
- HSyncSearchPos
- HSyncInsertPos
- VSyncSearchPos
- VSyncInsertPos
- SyncMaster
- BlackLevel
- WhiteLevel

For an explanation of these fields, see Table 1

6 The FGI functions

6.1 Basic functions

The user of this interface can do the following with the basic functions:

- **Open** or re-open a DT frame grabber device.
- **Close** a DT framegrabber device.
- **Grab** an image from the DT frame grabber device.
- **Get** a parameter of the DT frame grabber device.
- **Set** a parameter of the DT frame grabber device.

These functions are DLL's that can be executed in the Matlab workspace by just typing the DLL name with, if necessary, some input arguments.

6.1.1 Open a DT frame grabber device

DLL name openfg.dll

Syntax devinfo = openfg('alias')

Description This function will let the user open a DT frame grabber device. The user has to give the frame grabbers alias as input argument. The user already knows this alias. If the device is already open, a warning will be printed and the devinfo structure will be returned.

Input The *alias* (alias) of the frame grabber device.

Remarks It will be verified if the input is a valid string. If it's valid, it will try to open the frame grabber device with that specific alias. If more than one frame grabber is installed in the computer, it is necessary to give unique aliases to these devices (see Table 1). If the frame grabber is already open, it will give a warning message and return the handle of the device.

Output If the input isn't valid or the frame grabber device can't be opened for some reason, some error message will be shown to the user. If everything's ok, this function will return a structure with the *device information* (devinfo) of the opened frame grabber device.

Example To open the frame grabber with alias 'DT3152' and store the device info structure in the variable *m*, type in the Matlab Command Window:

```
m = openfg('DT3152');
```

6.1.2 Close a DT frame grabber device

DLL name closefg.dll

Syntax closefg(devinfo) or closefg('alias');

Description This function will let the user close an opened DT frame grabber device. The user has to give the frame grabbers device information structure as input argument. Executing the *openfg* function has retrieved this device information structure

Input The *device information structure (devinfo)* or the alias of an opened frame grabber.

Remarks It will be verified if the input is a valid device information structure. If it's valid, it will try to get the *deviceid* from the device information structure.

Output If the input isn't valid or the frame grabber can't be closed for some reason, some error message will be shown to the user. If everything's ok, it will close the opened frame grabber device.

Example 1: To close a frame grabber with its device id stored in the variable *m* ,type in the Matlab Command Window :

```
closefg(m);
```

Example 2: To close a frame grabber with the alias 'DT3155', type in the Matlab Command Window:

```
closefg('DT3155');
```

6.1.3 Grab an image from the DT frame grabber device

DLL name grabfg.dll

Syntax
- imgmatrix = grabfg(devinfo)
- moviematrix = grabfg(devinfo, imgmatrix)
- [imgmatrix, moviematrix] = grabfg(devinfo)
- grabfg(devinfo, imgmatrix)
- grabfg(devinfo, imgmatrix, moviematrix)
- grabfg(devinfo, imgmatrix, moviematrix, scaling factor)

Description This function will let the user grab an image from an opened frame grabber device. The user has to give the frame grabbers device information structure (handle) as input argument. When used with the moviematrix argument, a standard bitmap will be

returned according to the matlab movie format. The scaling factor, can be used for scaling down this bitmap.

Input The *device information structure (devinfo)* of an opened frame grabber device and optionally the pre-allocated image matrix (imgmatrix), a movie matrix (moviematrix) and a scaling factor for movies. The movie matrix has size [m, 1] and consists of doubles. Calculate m as follows: m = 388 + (img height / scaling factor) * (img width / scaling factor) / 8. The movie scaling factor has 2 as a default value.

Remarks It will be verified if the input is a valid device information structure. If it's valid, it will try to get the *deviceid* from the device information structure. If the second input is given, the function will check the size of the second input and checks if there's enough room to store the grabbed image. If there's enough room the variable (of type imgmatrix) will be overwritten without creating a new matrix, else a new matrix will be created and stored in that variable.

Output If the input isn't valid or the frame grabber isn't able to grab an image, some error message will be shown to the user. If everything's ok it will return the grabbed image stored in a *Matlab matrix* (imgmatrix). The image is stored in transposed way. To show the image correctly in Matlab you can use the accent ' on the imgmatrix to correct it (=> image(imgmatrix'))

Example There are two ways to use this function. Both will be shown here. Variable *m* is the device info structure, which was returned from the function *openfg*.

1. Just grab an image and store it in the variable *img*, type:

```
im = grabfg(m);
```

To show this image in Matlab (in the right way, not transposed) type:

```
image(im');
```

2. First create a matrix of the appropriate size and type where the image will be stored:

```
im = uint8(zeros(768,576));
```

Then grab the image:

```
grabfg(m,im);
```

Show the image in Matlab transposed:

```
image(im);
```

6.1.4 Get a parameter from the DT frame grabber device

DLL name getfg.dll
Syntax parvalue = getfg(devinfo, 'parname');

Description This function will let the user get a parameter of DT frame grabber device.

Input he *device information structure (devinfo)* of the opened frame grabber device and the *parameter name* (parname).

Remarks It will be verified if the first input argument is a valid device information structure and if the second input argument is a parameter name. If it's valid, it will try to get the device id from the device information structure.

The *parameter name* (parname) can be:

- 'DeviceName'
- 'DeviceId'
- 'InputChannel'
- 'PixelDepth'
- 'TimeOut'
- 'InputFilter'
- 'FrameType'
- 'FrameSize' (the return value is a structure with frame size info)
- 'FrameTop'
- 'FrameLeft'
- 'FrameWidth'
- 'FrameHeight'
- 'HorFrameInc'
- 'VerFrameInc'
- 'VideoSize' (the return value is a structure with video size info)
- 'TotalPixPerLine'
- 'BackPorchStart'
- 'ClampStart'
- 'ClampEnd'
- 'FirstActivePixel'
- 'ActivePixelCount'
- 'TotalLinesPerFld'
- 'FirstActiveLine'
- 'ActiveLineCount'
- 'ClockFreq'
- 'ExtOnLoToHi'
- 'VideoType'
- 'CSyncTresh'
- 'CSyncSource'

- 'LineOnLoToHi'
- 'FieldOnLoToHi'
- 'SyncSentinel'
- 'HSyncSearchPos'
- 'HSyncInsertPos'
- 'VSyncSearchPos'
- 'VSyncInsertPos'
- 'SyncValue'
- 'BlackLevel'
- 'WhiteLevel'

For an explanation of these parameters, see appendix table 1

See appendix table 2 for the allowance/availability to get a parameter from the selected frame grabber.

Output If the input isn't valid or the frame grabber can't get the parameter value of the chosen parameter name, some error message will be shown to the user. If everything's ok, it will return the *parameter value* (parvalue) of the chosen parameter name of the device.

Example To get the black level that will be stored in the variable *bl*, and assuming that the variable *m* stores the device info of the current frame grabber, type:

```
bl = getfg(m,'BlackLevel');
```

6.1.5 Set a parameter of the DT frame grabber device

DLL name setfg.dll

Syntax setfg(devinfo, 'parname', parvalue); or setfg(devinfo);

Description This function will let the user set a parameter of a DT frame grabber device. Setfg(m) lists all the possible variables.

Input The *device information structure (devinfo)* of the opened frame grabber device, the *parameter name* (parname) of the parameter that has to be set and the *parameter value* (parvalue) of that parameter.

Remarks It will be verified if the first input argument is a valid device information structure and if the second input argument is a parameter name. If it's valid, it will try to get the device id from the device information structure. To get the updated device info structure when you have set a parameter, use `getfg(devinfo);`
Setfg(m) gives a listing of all possible values.
The *parameter name* (parname) can be:

- 'InputChannel'
- 'PixelDepth'

- ‘TimeOut’
- ‘InputFilter’
- ‘FrameType’
- ‘FrameSize’
- ‘FrameTop’
- ‘FrameLeft’
- ‘FrameWidth’
- ‘FrameHeight’
- ‘HorFrameInc’
- ‘VerFrameInc’
- ‘VideoSize’
- ‘TotalPixPerLine’
- ‘BackPorchStart’
- ‘ClampStart’
- ‘ClampEnd’
- ‘FirstActivePixel’
- ‘ActivePixelCount’
- ‘TotalLinesPerFld’
- ‘FirstActiveLine’
- ‘ActiveLineCount’
- ‘ClockFreq’
- ‘ExtOnLoToHi’
- ‘VideoType’
- ‘CSyncTresh’
- ‘CSyncSource’
- ‘LineOnLoToHi’
- ‘FieldOnLoToHi’
- ‘SyncSentinel’
- ‘HSyncSearchPos’
- ‘HSyncInsertPos’
- ‘VSyncSearchPos’
- ‘VSyncInsertPos’
- ‘SyncMaster’
- ‘SyncValue’
- ‘BlackLevel’
- ‘WhiteLevel’
- ‘ExposureMode’
- ‘ExposureValue’
- ‘CameraType’
- ‘DigitalIO’
- ‘DigitalIOConfig’

For an explanation of these parameters, see Table 1.
See Table 2 for the possible parvalues and allowance/availability to set a parameter of the selected frame grabber.

Output If the input isn't valid or the frame grabber can't be set with the chosen parameter value, some error message will be shown to the user. If everything's ok, it will set the chosen parameter of the frame grabber device to the chosen parameter value.

Example To set the frame height to 400 pixels, and assuming that the variable *m* stores the device info of the current frame grabber, type:

```
setfg(m,'frameheight',400);
```

6.2 Advanced functions

The user of this interface can do the following with the advanced functions:

- **Reset** a DT frame grabber device.
- **Load** settings for a camera from a file.
- **Check** whether the frame grabber has been opened.
- **Recover** the handle of the opened device when the handle has been lost.

These functions are DLL's that can be executed from the Matlab workspace by just typing the DLL name with, if necessary, some input arguments.

6.2.1 Reset a DT frame grabber device

DLL name resetfg.dll

Syntax resetfg(devinfo);

Description This function will let the user reset an opened DT frame grabber device. The user has to give the frame grabbers device information structure as input argument. Executing the *openfg* function has retrieved this device information structure

Input The *device information structure (devinfo)* of an opened frame grabber device.

Remarks It will be verified if the input is a valid device information structure. If it's valid, it will try to get the *deviceid* from the device information structure.

Output If the input isn't valid, some error message will be shown to the user. If everything's ok, it will reset the opened frame grabber device.

Example To reset a frame grabber with its device id stored in the variable *m* ,type in the Matlab Command Window :

```
resetfg(m);
```

6.2.2 Load settings for a camera from a file

DLL name loadfg.dll

Syntax loadfg(devinfo,filename);

Description This function will let the user load settings from a file. The user has to give the frame grabbers device information structure as input argument, as well as the filename (with full path). Executing the *openfg* function has retrieved this device information structure

Input The *device information structure (devinfo)* of an opened frame grabber device, the filename (with full path).

Remarks It will be verified if the input is a valid device information structure. If it's valid, it will try to get the *deviceid* from the device information structure.

Output If the input isn't valid, some error message will be shown to the user. If everything's ok, it will load the settings from the file into the frame grabber. To see the new settings, use the *getfg* function.

Example To load the settings from the file *pulnix.ini* with path c:\camfiles and assuming that the variable *m* stores the device info of the current frame grabber, type:

```
loadfg(m,'c:\camfiles\pulnix.ini');
```

The structure of the setting file is like a standard Windows '95 ini-file.

It looks like this :
(The full example can be found in the /camfiles directory of the FGI Dev. Kit)

```
[Acquire Setup Parameters]
Alias=DT3155
InputSource=0
TimeOut=10

[Channel 0]
BlackLevel=4000
WhiteLevel=500000
InputFilter=AcNone
FrameType=FrameOdd
VideoType=Composite

[Channel 1]
BlackLevel=4000
WhiteLevel=500000
InputFilter=AcNone
FrameType=FrameEven
VideoType=Composite

[Channel 2]
BlackLevel=4000
WhiteLevel=500000
InputFilter=AcNone
FrameType=FrameEven
VideoType=Composite
```

6.2.3 Check the status of a DT framegrabber device

DLL name isopenfg.dll

Syntax isopenfg('alias');

Description This function will let the user check whether the frame grabber device has been opened or not. It will return an integer.

Input The *alias* (alias) of the frame grabber device.

Remarks It will be verified if the input is a string. If it's valid, the function will return a double: 1 when open, 0 when closed.

Output If the input isn't valid, some error message will be shown to the user. If everything's ok, the function will return an integer indicating the status of frame grabber device. If the function returns 1 the frame grabber has been opened, otherwise the function returns zero.

Example To check if a frame grabber has been opened or not, type in the Matlab Command Window :

```
isopenfg('DT3155');
```

6.2.4 Returning the handle of a DT frame grabber device

DLL name gethandlefg.dll

Syntax gethandlefg('alias');

Description This function will return the handle of the frame grabber device when you lost the handle to the device.

Input The *alias* (alias) of the frame grabber device.

Remarks It will be verified if the input is a string. If it's valid string, the function will return the device information structure of the device.

Output If the input isn't valid, some error message will be shown to the user. If everything's ok, the function will return a device information structure of the opened frame grabber device.

Example To regain the handle to the frame grabber, type in the Matlab Command Window:

```
m=gethandlefg('DT3155');
```

7. Troubleshooting

Question
If I set a parameter, why does the devinfo structure that was retrieved when opening the device, not display the new value?

Answer
When you set a parameter you have to update the structure by yourself if you want. Use *getfg(devinfo)* to do that.

Question
If I execute a function, nothing happens and I have to wait for about 10 seconds until I can continue with running functions in the Matlab Command Window.

Answer
The timeout of the frame grabber is set at an initial value of 10 seconds. So the device will try to execute its functions for 10 seconds. After these 10 seconds it will stop and returns to the Matlab function. You can make this time out shorter by changing the time out parameter with the *setfg* function

Question
If I grab an image and show this in Matlab with the image function it won't be shown correctly, i.e. it is transposed. How can I correct this?

Answer
If an image is grabbed in Matlab it will be stored column wise into a Matlab matrix. The image will be shown transposed if you just *type image(img)*, where *img* is the matrix where you stored the image. To correct this type *image(img')*. So the transposed image will be transposed again and it will be correctly shown in Matlab.

Question
If I use *loadfg(devinfo,'setting.ini')* the values didn't change at all (when calling *getfg(devinfo))*. What happened?

Answer
You have to type in the full pathname and filename.

PART II: Technical Guide

7 Introduction

The technical guide is mainly meant for developers who want to enhance or make changes to this Matlab interface. In this guide we will describe what we've included in the Matlab interface for DT Mach Frame grabbers Development Kit.

D. J. Oei and A Kapoerchan have written the original technical guide in 1998. This technical guide has been updated en partly rewritten by R. Heil and C. L. Wauters, June 2000 and by M.A.E.Bakker and L.I.Oei, July 2002 after bug-fixing the Matlab FGI source code.

Some knowledge about frame grabbers is required to understand some source code that is shown in this guide. The FG SDK manual should help you to understand the source code.

The latest revision of FGI was tested using Matlab 5.3, the environment and compiler of MS Visual C++ 6.0 was used for building the program.

For a complete explanation of the changes made in 2000, we refer to the *FGI Report 2000, R. Heil, and C.L. Wauters*

In the next chapter we will explain the structure of the directories that are included in the Matlab interface for DT FG Development Kit. After that we will discuss the compiling and linking of the MEX-Files. In the last chapter there will be some parts of the source codes explained, so a global view of the interface code structure can be gained.

8 The necessary programs and files

The Matlab interface for DT Mach Frame grabbers Development Kit is divided in several directories. They will be described below:

/bin - **DLL's and M-files**
This directory includes the M-files and DLL-files of the Matlab interface for DT Mach Frame grabbers.

/camfiles - **Example setting files**
This directory includes an example setting file for the Pulnix camera working on the frame grabber DT3152

/compile - **Developer's files**
This directory includes the following files for developing and changing the Mex-files. Some hard drive letters in the files should be changed if necessary.

/demos - **Some FGI demos**
This directory includes some demos to use with some frame grabbers.

/docs - **User's & Technical Guide**
This directory includes the specification and the User Manual & technical guide of the FGI.

/src - **Source code of the FGI**
This directory includes the source code of the FGI.

In every directory a readme.txt is included which will describe the contents of the appropriate directory.

9 Compiling and Linking

We made an M-file to have a possibility for compiling and linking the MEX source files easily. In the first part of the M-file we define the default locations for some include files and libraries.

At the time we wrote this document the following directories were standard:

General include files	:	y:\software\dt\fgsdk32\include
DT3152 include files	:	y:\software\dt\fgsdk32\include\dt3152
DT3157 include files	:	y:\software\dt\fgsdk32\include\dt3157
ColorSDK include files	:	y:\software\dt\fgsdk32\include\colorsdk
Open Layer Libraries	:	y:\software\dt\fgsk32\lib\olimg32.lib and olfg32.lib

If you run the M-file, it will first ask you for the “sourcedir”, this is the directory were the MEX source files exist. After that you will have to give the “output” directory, this is the directory were the Dynamic Link Libraries are placed after compilation and linkage.

Below you will find the source code of buildall.m:

```
%
%M-file for compiling and linking the FGI source files
%
%Date: 01-06-2000
%Copyright (c) 2000 R.Heil, C. L. Wauters
%SYNTAX buildall;
%
%
%DESCRIPTION File for easily compiling and linking the FGI source files.
%
%INPUT The user is asked to input the source directory and output directory
%
%EXAMPLE:
%buildall;
%

INC = '-Iy:\software\dt\fgsdk32\include';
INC52='-Iy:\software\dt\fgsdk32\include\dt3152';
INC57='-Iy:\software\dt\fgsdk32\include\dt3157';
INCCO='-Iy:\software\dt\fgsdk32\include\colorsdk';
LIB = 'y:\software\dt\fgsdk32\lib\olimg32.lib y:\software\dt\fgsdk32\lib\olfg32.lib';

disp(' ');
%SOURCE = input('Enter the source directory (example: g:\\sourcedir ) -> ','s');
SOURCE = '../Src';
%OUTDIR = input('Enter the output directory (example: g:\\matlab ) -> ','s');
OUTDIR = '../Bin';
SOURCE = strcat(SOURCE,'\\');

disp(' ');
disp('Building FGI source files');
disp(' ');

disp('Compiling setfg');
com = strcat(SOURCE,'setfg.c');
```

```
mex(com,'-O',INC,INC52,INC57,INCCO,LIB,'-outdir',OUTDIR)

disp('Compiling getfg');
com = strcat(SOURCE,'getfg.c');
mex(com,'-O',INC,INC52,INC57,INCCO,LIB,'-outdir',OUTDIR)

disp('Compiling loadfg');
com = strcat(SOURCE,'loadfg.c');
mex(com,'-O',INC,INC52,INC57,INCCO,LIB,'-outdir',OUTDIR)

disp('Compiling resetfg');
com = strcat(SOURCE,'resetfg.c');
mex(com,'-O',INC,INC52,INC57,INCCO,LIB,'-outdir',OUTDIR)

disp('Compiling grabfg');
com = strcat(SOURCE,'grabfg.c');
mex(com,'-O',INC,INC52,INC57,INCCO,LIB,'-outdir',OUTDIR)

disp('Compiling isopenfg');
com = strcat(SOURCE,'isopenfg.c');
mex(com,'-O',INC,INC52,INC57,INCCO,'-outdir',OUTDIR)

disp('Compiling gethandlefg');
com = strcat(SOURCE,'gethandlefg.c');
mex(com,'-O',INC,INC52,INC57,INCCO,'-outdir',OUTDIR)

disp('Compiling openfg');
com = strcat(SOURCE,'openfg.c');
mex(com,'-O',INC,INC52,INC57,INCCO,LIB,'-outdir',OUTDIR)

disp('Compiling closefg');
com = strcat(SOURCE,'closefg.c');
mex(com,'-O',INC,INC52,INC57,INCCO,LIB,'-outdir',OUTDIR)

disp('Compiling mainfg');
com = strcat(SOURCE,'mainfg.c');
mex(com,'-O',INC,INC52,INC57,INCCO,'-outdir',OUTDIR)
```

code example 2. Source buildall.m

10 Source Codes

In this chapter we describe the differences between the old FGI (1998) and the new FGI (2000). We also explain some part of the changed source code.

10.1 Code optimization

First we will give an explanation of the Open Layer calls and the error return codes.

The FGI MEX functions make use of the frame grabber Open Layer functions; these function calls will normally return an "ok" code. If the function does not succeed it will return an error code.
In the MEX files a switch statement catches this return code. If it was an error message, Matlab will display an appropriate message in the Matlab Command Window.

The error handling in the MEX files is done by the MEX function *"mexErrMsgTxt"*. This function will destroy all existing matrices made by the MEX file and it will return control to the Matlab environment.

Another way of exception handling is the MEX function called *"mexWarnMsgTxt"*. This function will display a warning message in the Matlab Command Window, but the MEX file will keep control and Matlab will continue executing the MEX file.

We discovered that in a lot of the Matlab functions there were same kinds of case statements.

An example from *openfg*:

```
*********************************************************************************************

/* Framesize info: Get the default vertical frame inc */
    result = OlFgQueryInputControlValue(
                      DeviceId, InputSource,
                      OLC_FG_CTL_VER_FRAME_INC, &VerFrameInc);
    switch (result)
     {
      case OLC_STS_NORMAL:
       break;
      case OLC_STS_NULL:
       mexPrintf("Pointer supplied is NULL.\n");
       mexErrMsgTxt("Unable to get the vertical frame inc.\n");
       return;
      case OLC_STS_CONTROL:
       mexPrintf("Undefined input control.\n");
       mexErrMsgTxt("Unable to get the vertical frame inc.\n");
       return;
      case OLC_STS_INPUTSOURCE:
       mexPrintf("Illegal inputsource supplied.\n");
       mexErrMsgTxt("Unable to get the vertical frame inc.\n");
       return;
      case OLC_STS_NONOLMSG:
       mexPrintf("Unsupported or unknown message passed.\n");
       mexErrMsgTxt("Unable to get the vertical frame inc.\n");
       return;
      default:
       mexPrintf("Unknown error.\n");
       mexErrMsgTxt("Unable to get the vertical frame inc.\n");
       return;
       }
*********************************************************************************************
```

code example 3. Switch openfg

In the *openfg* source code we found a variant of code example 3. In this version the value of a variable will be set when the SDK returns an OLC_STS_CONTROL state (in this example *FirstActivePixel = -1*). This means that the setting is not supported.

Important in this switch is that the OLC_STS_NORMAL and the OLC_STS_CONTROL state break. This means that the MEX file still continues because there was no return statement like the other cases.

The variant of code example 3 in *openfg* will be shown:

```
*********************************************************************************************

/* Videosize info: Get the default value of the First Active Pixel */

result = OlFgQueryInputControlValue(DeviceId, inputSource, OLC_FG_CTL_FIRST_ACTIVE_PIXEL, &FirstActivePixel);

switch (result)
     {
      case OLC_STS_NORMAL:
       break;
      case OLC_STS_NULL:
       mexPrintf("Pointer supplied is NULL.\n");
       mexErrMsgTxt("Unable to get the First Active Pixel.\n");
       return;
      case OLC_STS_CONTROL:
       //mexPrintf("Undefined input control.\n");
       //mexWarnMsgTxt("Unable to get the First Active Pixel.\n");
       FirstActivePixel = -1;
       break;
      case OLC_STS_INPUTSOURCE:
       mexPrintf("Illegal inputsource supplied.\n");
       mexErrMsgTxt("Unable to get the First Active Pixel.\n");
       return;
      case OLC_STS_NONOLMSG:
       mexPrintf("Unsupported or unknown message passed.\n");
       mexErrMsgTxt("Unable to get the First Active Pixel.\n");
       return;
      default:
       mexPrintf("Unknown error.\n");
       mexErrMsgTxt("Unable to get the First Active Pixel.\n");
       return;
     }

*********************************************************************************************
```

code example 4. Variant switch openfg

We noticed in the old FGI source that the return statements after the mexErrMsgTxt are obsolete, because the Matlab environment regains control after a mexErrorMsgTxt. So it will never execute the return code.

To increase the readability of the code we decided to make an overall C-file (*check.c*) where the case statement will be executed with the proper parameter.
The error message, which will be shown in the Matlab Command Window, will be: “Unable to get the <*parameter*>”. For example “Unable to get the FrameType”

A part of *check.c* will be shown:

```
*********************************************************************************************
void DEF_checkError(OLT_APISTATUS result, char error[30])
{
       char errorStr[80];
       strcpy(errorStr, "Unable to get the ");
```

```
       strncat(errorStr, error, 30);
       switch (result)
         {
              case OLC_STS_NORMAL:break;
              case OLC_STS_NULL:
               mexPrintf("Pointer supplied is NULL.\n");
               mexErrMsgTxt(errorStr);
              case OLC_STS_CONTROL:
               mexPrintf("Undefined input control.\n");
               mexErrMsgTxt(errorStr);
              case OLC_STS_INPUTSOURCE:
               mexPrintf("Illegal inputsource supplied.\n");
               mexErrMsgTxt(errorStr);
              case OLC_STS_NONOLMSG:
               mexPrintf("Unsupported or unknown message passed.\n");
               mexErrMsgTxt(errorStr);
              default:
               mexPrintf("Unknown error.\n");
               mexErrMsgTxt(errorStr);
         }
}

void DEF_checkError_Control(OLT_APISTATUS result, char error[30]) {
       char errorStr[80];
       strcpy(errorStr, "Unable to get the ");
       strncat(errorStr, error, 30);

       switch (result)
         {
              case OLC_STS_NORMAL: break;
              case OLC_STS_CONTROL: break;
              case OLC_STS_NULL:
               mexPrintf("Pointer supplied is NULL.\n");
               mexErrMsgTxt(errorStr);
              case OLC_STS_INPUTSOURCE:
               mexPrintf("Illegal inputsource supplied.\n");
               mexErrMsgTxt(errorStr);
              case OLC_STS_NONOLMSG:
               mexPrintf("Unsupported or unknown message passed.\n");
               mexErrMsgTxt(errorStr);
              default:
               mexPrintf("Unknown error.\n");
               mexErrMsgTxt(errorStr);
         }
}

*********************************************************************************************
```

code example 5. Errorcheck functions check.c

In the new source code the case statement (see code example 3) is replaced by a call to the *DEF_checkError (check.c)* function (see code example 6) with the parameter “vertical frame inc.”.

```
*********************************************************************************************
/* Framesize info: Get the default vertical frame inc */

result = OlFgQueryInputControlValue( DeviceId, InputSource, OLC_FG_CTL_VER_FRAME_INC, &VerFrameInc);
DEF_checkError(result, "vertical frame inc.");

*********************************************************************************************
```

code example 6. Call DEF_checkError check.c

In the old switch (see code example 3) the case of OLC_STS_NORMAL breaks and the MEX function continues execution. The other cases return (because of the mexErrMsgTxt function call) and exit directly the MEX function returning the control to Matlab. The same functionality is obtained in check.c

figure1 illustrates the new situation:

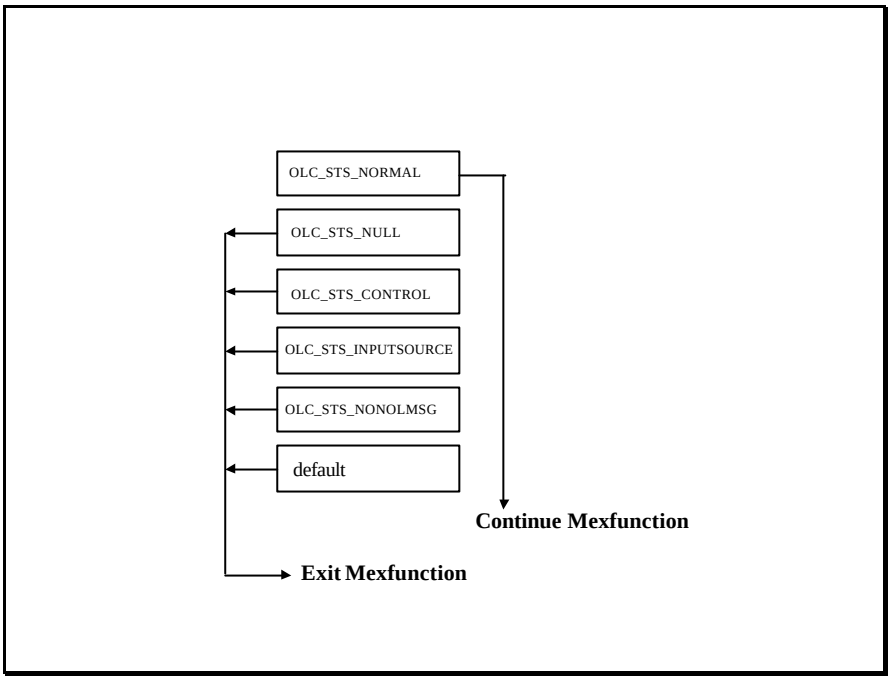

figure1. Error handling DEF_checkError schematically

The main issue of code example 4 is that in the case of OLC_STS_CONTROL a variable (in this example *FirstActivePixel = -1*) will be set and the MEX function continues.

The new function call in *openfg* will assign the value of the variable in the case of OLC_STS_CONTROL. The other states (except OLC_STS_NORMAL) return and exit the MEX function directly (because of the mexErrMsgTxt function call).

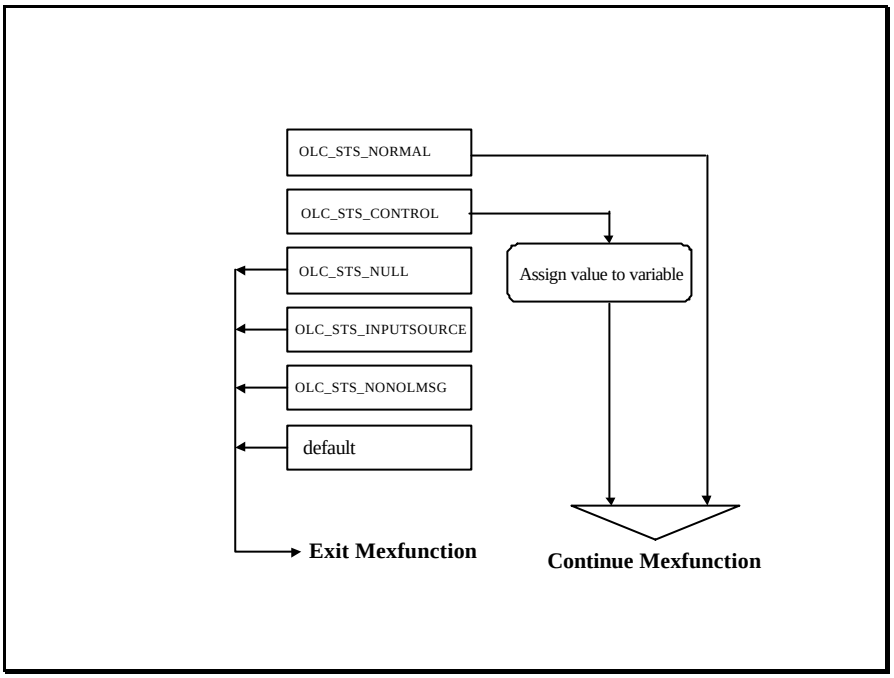

figure2. Error handling DEF_checkError_Control schematically

The new function call in *openfg* will be instead of code example 4:

```
*********************************************************************************************

/* Videosize info: Get the default value of the First Active Pixel */

result = OlFgQueryInputControlValue( DeviceId, InputSource, OLC_FG_CTL_FIRST_ACTIVE_PIXEL,
&FirstActivePixel);

 DEF_checkError_Control(result, "First Active Pixel.");
 if (result == OLC_STS_CONTROL) FirstActivePixel = -1;

*********************************************************************************************
```

code example 7. New function call openfg

10.2 MainFG:

To solve a few FGI problems we need a structure containing information about the open devices. With this open devices list the *functions* are able to check whether a device can be opened, closed, etc. One problem is that it is not easy to share a structure between the different dynamic link libraries. The solution we proposed for this problem is to create a main function, which will own the device list; the other functions should then be able to communicate with this main function for retrieving and storing global data. This main function will be accessible from the Matlab interface; it is even possible to manipulate the open device list if necessary.

The *openfg* dynamic link library has five functions for manipulating the open devices list.
These functions are:

- **Add**: adds a device to the open device list, this function is mainly used in the *openfg* function.
  Syntax: mainfg('add', deviceName <string>, deviceHandle <integer>);
  Return value: <none>

- **Remove**: removes a device from the open device list. It is used in the *closefg* function.
  Syntax: mainfg('remove', deviceName <string>);
  Return value: <none>

- **Check**: is able to check if a device is open or not. This function could be used for example in *openfg* to check if the device is already open or not.
  Syntax: mainfg('check', deviceName <string>);
  Return value: "Yes" or "No"

- **Get**: returns the device handle for the given device name.
  Syntax: mainfg('get', deviceName <string>);
  Return value: device handle <integer>

- **Print**: built-in for testing purposes, it will show a listing of the open devices.
  Syntax: mainfg('print');
  Return value: it will print the open devices list.

We will give an example how the *mainfg* stores the *alias* name and the device handle (*deviceId*) in the main structure.

For this example we assume that there are three PCI frame grabber cards are installed in the computer. The DT3133 frame grabber card has got three aliases, DT3133-1, DT3133-2 and DT3133-3.

```
mainfg('print')

result:
Device nr: 0 --> DT3157, 1
Device nr: 1 --> DT3152, 2
Device nr: 2 --> DT3133-1, 3
Device nr: 3 --> DT3133-2, 4
```

code example 8. Mainfg('print')

Before using the get, remove or add task it is necessary to use *checkDevice* first to verify if the device is open or not. In the following paragraph we will explain how we use the *mainfg* function.

10.3 CheckDevice:

Because most functions are using the check task, we have decided to make the checkDevice function. This function will accept an mxArray that must contain the device name. It will return true if the device is in the open devices list, otherwise it will return false.

```
*********************************************************************************************
bool checkDevice(mxArray *name[])
{
        mxArray *rhs[2], *lhs[1];
        int buflen;
        char *buffer = "";

        // Create rhs for checking deviceName
        rhs[0] = mxCreateString("check");
        rhs[1] = name[0];
        mexCallMATLAB(1, lhs, 2, rhs , "mainfg");

        // Read the answer (Yes or No)
        buflen = (mxGetM(lhs[0])*mxGetN(lhs[0])*sizeof(mxChar)) + 1;
        buffer = mxCalloc(buflen,sizeof(char));
        mxGetString(lhs[0],buffer, buflen);

        // Clear used memory
        mxDestroyArray(rhs[0]);
        mxDestroyArray(lhs[0]);

        return (strcmp(buffer, "Yes") == 0);
}
*********************************************************************************************
```

code example 9. Function checkdevice

Before calling *MexCallMATLAB(nr output, output, nr input, input)* one has to create some mxArrays. In this case we set the task to "check" by assigning the mxCreateString("check") to the first argument of *mainfg.* The second argument for mainfg will contain the device name which we would like to check, this is the device name which is given to the checkDevice function.
Now we are ready to call the *MexCallMATLAB* function, it will return the *mainfg* output to *lhs*. We are then able to read the answer string from *lhs[0]* by using the mxGetString function. Finally, we compare this answer to "Yes" or "No" and we will return the Boolean equivalent of it.

10.4 Grabfg.c

This function checks first if the inputted variable is already of the specified size and type. If it is, then this variable will be overwritten. If it isn't, a new image Matlab matrix will be created of the specified size (frameheight & framewidth) and type to store the acquired image.

We will give a short part of this code here:

```
if (((mxIsClass(output,"uint8")) && ((ULNG)mxGetM(output) == Width)
            && ((ULNG)mxGetN(output) == Height) && (PixelDepth == 1)) ||
 ((mxIsClass(output,"uint16")) && ((ULNG)mxGetM(output) == Width)
```

```
            && ((ULNG)mxGetN(output) == Height) && (PixelDepth == 2)) ||
 ((mxIsClass(output,"uint32")) && ((ULNG)mxGetM(output) == Width)
            && ((ULNG)mxGetN(output) == Height) && (PixelDepth == 4)))
 {
       p = mxGetPr(output);

       /* Grab an image from the framegrabber */
       result = OlFgAcquireFrameToHost((int)d, FrameId, p, Height*Width*PixelDepth);
       switch (result)
       {
         case OLC_STS_NORMAL:       //mexPrintf("Image has been grabbed\n");
                                    mexPutArray(output,"caller");
                                    OlFgDestroyFrame((int)d, FrameId);
                                    break;
         case OLC_STS_INVALIDFRAMEHANDLE:mexPrintf("Invalid Framehandle.\n");
                                    mexErrMsgTxt("Unable to grab an image.\n");
                                    return;
         case OLC_STS_BUFSIZ:       mexPrintf("Buffersize is too small.\n");
                                    mexErrMsgTxt("Unable to grab an image.\n");
                                    return;
         case OLC_STS_NULL:         mexPrintf("Pointer supplied is NULL.\n");
                                    mexErrMsgTxt("Unable to grab an image.\n");
                                    return;
         default:                   mexPrintf("Unknown error.\n");
                                    mexErrMsgTxt("Unable to grab an image.\n");
                                    return;
       }
  }
```

The full source code can be found in the /src directory of the Matlab Interface For DT Mach Framegrabbers Development Kit.

10.5 Getfg.c

This functions returns a value of the parameter you requested. You can update the device info structure by using this function with as the only input argument your device info structure variable.
See the source code in /src.

10.6 Setfg.c

This function sets a parameter value of a given parameter. To get the updated frame grabber settings, use the function getfg.
A part of the code:

```
else if (!(stricmp("FrameLeft", ParName)))
        {
         /* Want to set the frame left */
         ParValueLeft = (ULNG)mxGetScalar(prhs[2]);

         result = OlFgSetInputControlValue(
                    (OLT_IMG_DEV_ID)d, InputSource,
                    OLC_FG_CTL_FRAME_LEFT,
                    ParValueLeft, &OldParValueLeft);
         SETFG_checkError(result, "frame left.");
       }
```

Explanation of the above code:
First the inputted parameter name will be compared with “frameleft”. If it’s equal it will get the inputted parameter value and this will be used too set the specified parameter of the frame grabber. If it’s set, then no error message will be given, else a specified error message will be displayed in Matlab.
See the complete source code in /src.

10.7 Resetfg.c

This function resets the frame grabber to its nominal state. To get the updated frame grabber settings, use the function getfg.
See the source code in /src.

10.8 Dynamic libraries

Problem Description: The dynamic libraries for all frame grabbers are always required. Usual situation is that a user only has one frame grabber card installed. Only the library of the frame grabber, which is attempted to be open, shall be required.

To solve the problem in the 1998 version we had to dynamically load the libraries instead of linking all device drivers to the compiled MEX-files. If it is necessary to use one of the specific device driver functions the dynamic link library will be automatically loaded. In this new situation it isn't necessary to install all the other frame grabber device drivers, only the device driver of the installed frame grabber board is required.

The following example source code shows us how to handle this dynamic loading:

```
*********************************************************************************************

DllHandle = LoadLibrary("DT315732.DLL");

if (DllHandle) {
Dt3157EnableSyncMasterMode = GetProcAddress(DllHandle,"Dt3157EnableSyncMasterMode");
}
*********************************************************************************************
```

code example 10.Dynamically loading openfg

First we open the dynamic link library (DLL) by calling the LoadLibrary function, if it succeeds the handle will be stored in the DllHandle variable. With GetProcAddress it is then possible to get a pointer to the required function, for example Dt3157EnableSyncMasterMode. After this initialization you can use the dynamically loaded function in the same way as a normal function.

10.9 Implementation Ini files

Because the old *loadfg* function (1998) didn't work at all, we have decided to make a new *loadfg* function. At this moment it is only possible to set the common settings, we didn't implement some extensions because of the extra programming overload. But it is possible to easily extend the *loadfg* function in the future for some frame grabber extensions.
Loadfg uses a camera initialization file (*.ini), this file will have this common form:

```
[Acquire Setup Parameters]
Alias=DT3155
InputSource=0
TimeOut=10

[Channel 0]
BlackLevel=4000
WhiteLevel=500000
InputFilter=AcNone
FrameType=FrameOdd
```

```
VideoType=Composite

[Channel 1]
BlackLevel=4000
WhiteLevel=500000
InputFilter=AcNone
FrameType=FrameEven
VideoType=Composite

[Channel 2]
BlackLevel=4000
WhiteLevel=500000
InputFilter=AcNone
FrameType=FrameEven
VideoType=Composite

[Channel 3]
BlackLevel=4000
WhiteLevel=500000
InputFilter=AcNone
FrameType=FrameEven
VideoType=Composite
```

code example 11.Ini file example DT3155

In the “Acquire Setup Parameters” section we will have to describe some settings common for all channels, these settings are “InputSource” and “TimeOut”.
In the “Channel #” sections we will put the settings for each of the existing channels. In this case it is the DT3155 Frame Grabber with four channels, so there are four sections named “Channel 0” through “Channel 3”. Each section contains specific settings for that channel.

The following settings can be used in the camera initialization file:

- InputSource;
- TimeOut;
- BlackLevel;
- WhiteLevel;
- TotalPixPerLine;
- BackPorchStart;
- ClampStart;
- ClampEnd;
- FirstActivePixel;
- TotalLinesPerFld;
- FirstActiveLine;
- ActiveLineCount;
- FrameLeft;
- FrameTop;
- FrameWidth;
- FrameHeight;
- HorFrameInc;
- VerFrameInc;
- InputFilter;
- FrameType;
- VideoType.

If one of these features is not supported it will just be ignored. In the case that it is supported, but if an illegal argument is given, it will throw a warning message. The *loadfg* function will continue and tries to set the other variables.

11 Changes July 2002

In July 2002, some small bugfixes were aplied in the process of studying the FGI to build a new FGI for different frame grabber models.

11.1 The 'clear all' bug

When typing 'clear all' or 'clear mex' from within Matlab, the open device data on frame grabbers that were still open was lost. Every time this occurred Matlab had to be closed and restarted to be able make use of the FGI functions again.

The solution to this was found in using the mexLock() function from within fgmain. The implementation of a separate fgmain function in the 2000 version made this easy solution possible. Below is a part of the source code of fgmain to show the changes.

```
*************************************************************************
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])

{
  int buflen;
  int i;
  double *y;
  char *task = "";
  char *name = "";
  bool found = false;

  /* Added by M.Bakker 2002/06/24 12:35
   * Use mexLock to prevent 'clear all' or 'clear mex' from clearing
   * mainfg.dll from Matlab memory.  The deviceArray stays intact.
   */
  mexLock();

  . . .
  . . .
  . . .
  . . .
*************************************************************************
```

11.2 Improvements in grabfg

The original complaint was that when an error was encountered, some of the device driver's resources were not properly released, resulting in a new error message and the need to close and re-open Matlab when the error occurred several times. A subroutine was added to properly release resources. A call to this subroutine is made every time before a call to mexErrMsgTxt in a place problems could occur. To make the source more readable, we had already changed the lay-out of the source code. After making the above mentioned changes we noticed the error messages after calling the OL SDK to acquire a frame were very undescriptive, so we added several new ones. While we were working on the source code and checking the documentation, we noticed that not all six (!) different syntaxes for calling grabfg were documented, so we fixed that as well. Below is some of the source code of grabfg to show the changes.

```
*************************************************************************

/*****************************************************************/
/** Filename: grabfg.c                                          **/
/** Authors: D.J. Oei, A. Kapoerchan                            **/
/** Date: 10 July 1998                                          **/
/** Last update by original authors: 17 August 1998             **/
/** Modified by: R.Heil, C.L.Wauters 17 April 2000              **/
/** Last update by R.Heil, C.L.Wauters: 01 June 2000            **/
/**                                                             **/
/** Modified by: L.I.Oei, M.A.E.Bakker                          **/
```

```
/** Updates: source cleaned up and properly indented 2002/07/02**/
/**          destroy frame on errorexit 2002/07/03              **/
/**          improved error messages after acquire frame 2002/07/04**/
/**                                                             **/
/*****************************************************************/
/** This C-file contains the source-code to grab an image.      **/
/*****************************************************************/
/** Syntax: imgmatrix = grabfg(devinfo)                        **/
/**         moviematrix = grabfg(devinfo, imgmatrix)           **/
/**         [imgmatrix, moviematrix] = grabfg(devinfo)         **/
/**         grabfg(devinfo, imgmatrix)                         **/
/**         grabfg(devinfo, imgmatrix, moviematrix)            **/
/**         grabfg(devinfo, imgmatrix, moviematrix, scaling factor)**/
/*****************************************************************/

#include <mex.h>
#include <stdlib.h>
#include "windows.h"
#include "olwintyp.h"
#include "olimgapi.h"
#include "olfgapi.h"
#include "Dtcolors.h"


/* Added by M.Bakker 2002/07/03 18:25
 * this subroutine calls OlFgDestroyFrame to release resources that were not
 * properly released when another error was encountered during the grabbing.
 */
void destroyFrameOnErrorExit(OLT_IMG_DEV_ID DeviceId, OLT_FG_FRAME_ID FrameId)
{
    OLT_APISTATUS result;
    result = OlFgDestroyFrame(DeviceId, FrameId);
    switch (result)
    {
       case OLC_STS_NORMAL: break;
       case OLC_STS_BUSY:   mexPrintf("Still busy with acquisition.\n");
                            mexPrintf("Failed to destroy allocated resources while recovering from the
                            break;
       case OLC_STS_INVALIDFRAMEHANDLE:  mexPrintf("Invalid frame handle.\n");
                                         mexPrintf("Failed to destroy allocated resources while
                                         break;
       case OLC_STS_FRAMENOTALLOCATED:   mexPrintf("Frame not allocated.\n");
                                         mexPrintf("Failed to destroy allocated resources while
                                         break;
       case OLC_STS_NONOLMSG:       mexPrintf("Unsupported or unknown message passed to the device
                                    mexPrintf("Failed to destroy allocated resources while recovering from
                                    break;
       case OLC_STS_FRAMEISMAPPED: mexPrintf("Cannot destroy a frame that is still mapped.\n");
                                    mexPrintf("Failed to destroy allocated resources while recovering from
                                    break;
       default:      mexPrintf("Unknown error.\n");
                     mexPrintf("Failed to destroy allocated resources while recovering from the error
                     break;
    }
    return;
}
. . .
. . .
. . .
case OLC_STS_BUSY:   destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07/03 */
                     mexPrintf("Acquisition section of the device still busy.\n");
                     mexErrMsgTxt("Unable to grab an image.\n");
                     return;
case OLC_STS_SYSERROR:destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07/03 */
                     mexPrintf("Unable to communicate with the device.\n");
                     mexErrMsgTxt("Unable to grab an image.\n");
                     return;
case OLC_STS_TIMEOUT:destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07/03 */
                     mexPrintf("Timeout waiting for acquisition.\n");
                     mexErrMsgTxt("Unable to grab an image.\n");
                     return;
case OLC_STS_FIFO_OVERFLOW: destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002
                            mexPrintf("Too much activity on host bus during transfer.\n");
                            mexErrMsgTxt("Unable to grab an image.\n");
                            return;
```

```
case OLC_STS_FIELD_OVERFLOW:destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002
                            mexPrintf("Too much activity waiting on host bus during transfer.\n");
                            mexErrMsgTxt("Unable to grab an image.\n");
                            return;
case OLC_STS_INVALIDFRAMEHANDLE:  destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker
                                  mexPrintf("Invalid Framehandle.\n");
                                  mexErrMsgTxt("Unable to grab an image.\n");
                                  return;
case OLC_STS_CLAMP:  destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07/03 */
                     mexPrintf("Clamp start exceeds clamp end.\n");
                     mexErrMsgTxt("Unable to grab an image.\n");
                     return;
case OLC_STS_VERTICALINC:  destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("Vertical increment not equal to 1.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_FIRSTACTPIX:  destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("First active pixel exceeds backporch start or clamp start.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_ACTPIXCOUNT:  destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("Active line area is not between backporch start and total pixels per
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_ACTLINECOUNT:  destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("Active line count exceeds total lines per field.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_FRAMELEFT:    destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker
                           mexPrintf("Frame left exceeds active pixel count.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_RANGE:  destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07/03 */
                     mexPrintf("Difference between whitelevel and blacklevel too large.\n");
                     mexErrMsgTxt("Unable to grab an image.\n");
                     return;
case OLC_STS_FRAMETOP:       destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("Frame top exceeds active line count.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_FRAMEWIDTH:   destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("Frame width not multiple of 4 or exceeds active pixel count.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_FRAMEHEIGHT:  destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("Frame height will cause digitalisation to exceed active pixel
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_HSYNCSEARCHPOS:destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("H sync search position precedes H sync insert position.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_VSYNCSEARCHPOS:destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("V sync search position precedes V sync insert position.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_INTERLACEDHGTGRAN:   destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker
                                  mexPrintf("Frame height granularity is illegal.\n");
                                  mexErrMsgTxt("Unable to grab an image.\n");
                                  return;
case OLC_STS_INTERLACEDTOPGRAN:   destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker
                                  mexPrintf("Frame top granularity is illegal.\n");
                                  mexErrMsgTxt("Unable to grab an image.\n");
                                  return;
case OLC_STS_WHITELEVEL:   destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("White level cannot be set.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_NONOLMSG:       destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("Unsupported or unknown message passed to the device driver.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
                           return;
case OLC_STS_MODECONFLICT:  destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                           mexPrintf("Mode conflict.\n");
                           mexErrMsgTxt("Unable to grab an image.\n");
```

```
                        return;
case OLC_STS_BUFSIZ:    destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07
                        mexPrintf("Buffersize is too small.\n");
                        mexErrMsgTxt("Unable to grab an image.\n");
                        return;
case OLC_STS_NULL:      destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker
                        mexPrintf("Pointer supplied is NULL.\n");
                        mexErrMsgTxt("Unable to grab an image.\n");
                        return;
default:            destroyFrameOnErrorExit((OLT_IMG_DEV_ID)DeviceId, FrameId); /* M.Bakker 2002/07/03 */
                    mexPrintf("Unknown error.\n");
                    mexErrMsgTxt("Unable to grab an image.\n");
                    return;
```

11.3 Comments on openfg

In the 2000 version, some Dutch comments are present, if this version is ever to be updated again by non-Dutch programmers some translations may come in handy:

"`// LET OP ER IS GEEN STS_CONTROL CODE, ZIE ONDERAAN`"
"attention, there is no STS_CONTROL code, see below"

"`// LET OP ER IS GEEN CONTROL CODE`"
"attention, there is no CONTROL code"

12 Appendices

Table 1: Description of the parameters of the device info structure.

Parname	Description
'DeviceName'	The name you assigned to the board when you configured the board's device driver.
'DeviceId'	The device identification of the opened Framegrabber device.
'InputChannel'	The currently selected video input channel.
'PixelDepth'	The pixeldepth of the frame of the Framegrabber board.
'TimeOut'	The time in seconds required for an action to occur before the operation times out.
'InputFilter'	The currently used hardware filter.
'FrameType'	The frame type you want to acquire when grabbing an image.
'FrameTop'	The first line of video, relative to the active video area, to digitize.
'FrameLeft'	The first pixel of video, relative to the active video area, to digitize.
'FrameWidth'	The number of pixels per line of video to digitize.
'FrameHeight'	The number of lines per field of video (or noninterlaced frame) to digitize.
'HorFrameInc'	The increment between adjacently sampled pixels (in pixels).
'VerFrameInc'	The increment between adjacently sampled lines (in lines).
'TotalPixPerLine	The total pixels per line, i.e. the no. of pixels in a single horizontal line of video.
'BackPorchStart'	The period between the horizontal sync signal and the start of the active video signal.
'ClampStart'	The start position indicating the beginning of an area (in pixels) where the Framegrabber clamps the video signal, as a pixel value offset from the beginning of the horizontal sync.
'ClampEnd'	The end position indicating the beginning of an area (in pixels) where the Framegrabber clamps the video signal, as a pixel value offset from the beginning of the horizontal sync.
'FirstActivePixel'	The beginning of the active video signal on the line, as a pixel value offset from the beginning of the horizontal sync.
'ActivePixelCount'	The number of pixels per line (in pixels) in the active video area.
'TotalLinesPerFld'	The number of lines in a single field of video, where a field is defined as a area between two consecutive vertical sync signals.
'FirstActiveLine'	The beginning of the active video signal within the field, as a line offset from the beginning of the vertical sync.
'ActiveLineCount'	The number of lines per field (or noninterlaced frame) in the active video area.
'VideoType'	Specifies the video type: Composite video or variable scan input.
'CsyncTresh'	The point at which the frame grabber acknowledges the sync signal.
'CSyncSource'	The video input channel on the board that is used as the composite sync source.
'LineOnLoToHi'	Line (horizontal) sync on a low-to-high transition
'FieldOnLoToHi'	Field (vertical) sync on a low-to high transition.

‘SyncSentinel’	This circuitry corrects for missing, extraneous, or low-level sync signals.
‘HSyncSearchPos’	The pixel location within a line at which the frame grabber begins to search for the horizontal sync.
‘HSyncInsertPos’	The position where the sync sentinel inserts a horizontal sync to synchronize to the video signal.
‘VSyncSearchPos’	The line location within a field at which the frame grabber begins to search for the vertical sync.
‘HSyncInsertPos’	The position where the sync sentinel inserts a vertical sync to stay in sync with the video signal
‘SyncMaster’	This uses the sync signals generated by the board to drive one or more camera’s
‘BlackLevel’	The voltage below which all other voltages are digitized to black.
‘WhiteLevel’	The voltage above which all other voltages are digitized to white.
‘Exposure’	Generates an exposure pulse to the camera if enabled.
‘ExposureValue’	Specifies the pulse duration and the logical polarity
‘CameraType	Specifies the current digital camera
‘DigitalIO’	The digital input from the input ports
‘DigitalIOConfig’	The digital input/output configuration

Table 2: Possible values of the parameters and their availability to set/get.

		getfg.dll				setfg.dll			
parname	**possible parvalues**	**52**	**53**	**55**	**57**	**52**	**53**	**55**	**57**
'alias'	DT3152, DT3153, DT3155, DT3157	**X**	**X**	**X**	**X**				
'devid'	>= 0	**X**	**X**	**X**	**X**				
'inputchannel'	0,1,2,3 (52); 0,1,2 (53); 0,1,2,3(55); 0 (57);	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
'pixeldepth'	1,2 or 4	**X**	**X**	**X**	**X**				
'timeout'	>= 0	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
'inputfilter'		**X**		**X**		**X**		**X**	
	(AC Coupled: no filter) = *AcNone*					**X**		**X**	
	(AC Coupled: 50 Hz filter) = *Ac50*					**X**		**X**	
	(AC Coupled: 60 Hz filter) = *Ac60*					**X**			
	(DC Coupled: no filter) = *DcNone*					**X**			
'frametype'		**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
	(Il. Frame: acq. on next even field) = *FrameEven*					**X**	**X**	**X**	
	(Il. Frame: acq. On next odd field) = *FrameOdd*					**X**	**X**	**X**	
	(Il. Frame: acq. on next field) = *FrameNext*					**X**	**X**	**X**	
	(Single field: acq. on next even field) = *FieldEven*								
	(Single field: acq. on next odd field) = *FieldOdd*								
	(Single field: acq. on next field) = *FieldNext*								
	(Non-Il. Frame – acq. on next frame) = *NonInterlaced*					**X**			**X**
'frametop'	0–4095 for 52 &57 (0–575 50Hz & 0-479 60Hz for DT3153)	**X**	**X**	**X**	**X**	**X**	**X**		**X**
'frameleft'	0–4095 for 52 &57 (0–767 50Hz & 0-636 60Hz for DT3153)	**X**	**X**	**X**	**X**	**X**	**X**		**X**
'framewidth'	4–4096 for 52 &57 (4–768 50Hz & 4-640 60Hz for DT3153)	**X**	**X**	**X**	**X**	**X**	**X**		**X**
'frameheight'	1–4095 for 52 &57 (0–576 50Hz & 0-480 60Hz for DT3153)	**X**	**X**	**X**	**X**	**X**	**X**		**X**
'horframeinc'	1-16	**X**	**X**	**X**	**X**	**X**	**X**		**X**
'verframeinc'	1-16	**X**	**X**	**X**	**X**	**X**	**X**		**X**
'totpix'	4-4096 (52 & 57)	**X**			**X**	**X**			**X**

‘backporch’	4-4095 (52)	X				X			
‘clampstart’	4-4095 (52)	X				X			
‘clampend’	4-4095 (52)	X				X			
‘firstpix’	4-4095 (52 & 57)	X			X	X			X
‘pixcount’	4-4096 (52 & 57)	X			X	X			X
‘totallines’	1-4096 (52 & 57)	X			X	X			X
‘firstline’	0-4095 (52 & 57)	X			X	X			X
‘linecount’	1-4096 (52 & 57)	X			X	X			X
‘pcmode’		X	X	X	X	X	X		X
	0 (Internal clock)	X	X	X	X	X	X		X
	1 (External clock)	X			X	X			X
‘pcvalue’		X	X	X	X	X			X
	Internal: 1000-20.000.000 Hz (52 & 57); Fixed (53)	X	X	X	X	X			X
	External: 0 (lo - hi off), 1 (lo - hi on)	X			X	X			X
‘videotype’		X	X	X	X	X			X
	(Composite video) = Composite	X	X			X			
	(Variable scan input) = VarScan	X			X	X			X
‘synctreshold’	-	X	X			X			
‘syncsource’	-	X	X			X			
‘lineonlotohi’	0 (disabled); 1 (enabled)	X			X	X			X
‘fieldonlotohi’	0 (disabled); 1 (enabled)	X			X	X			X

		getfg.dll				**setfg.dll**			
parname	**Possible parvalues**	**52**	**53**	**55**	**57**	**52**	**53**	**55**	**57**
‘syncsentinal’	(disabled) = Off (enabled) = On	X	X	X	X	X		X	X
‘hsyncsearchpos’	-	X	X	X	X	X			X
‘hsyncinsertpos’	-	X	X	X	X	X			X
‘vsyncsearchpos’	-	X	X	X	X	X			X
‘vsyncinsertpos’	-	X	X	X	X	X			X
‘syncmaster’	(disabled) = Off (enabled) = On	X				X			
‘blacklevel’	min: -1066800 max: 1075200 (52) min: 1275 max: 306000 (55)	X		X		X		X	
‘whitelevel’	min: -1066800 max: 3625200 (52) min: 351100 max: 1009000 (55)	X		X		X		X	
Extra for dt3157									
‘exposure’	0 (disabled); 1(enabled)								X

‘expvalue’	See manual of dt3157.				**X**				**X**
‘cameratype’					**X**				**X**
	16 (16 Bit Input)				**X**				**X**
	14 (14 Bit Input)				**X**				**X**
	12 (12 Bit Input)				**X**				**X**
	10 (10 Bit Input)				**X**				**X**
	81 (8 Bit Single Channel Input)				**X**				**X**
	82 (8 Bit Dual Channel Input)				**X**				**X**
‘digitalio’	See manual of dt3157.				**X**				**X**
‘digioconfig’	See manual of dt3157.				**X**				**X**

(Numbers 52, 53, 55 & 57 => DT31**), Il. = Interlaced, acq. = acquisition, **X** = available

Structure of the FGI